OOD Assignment 3 – 80 Points – UML/Java OO System Design and Development

This assignment is intended to be done by you as a solo effort. The goal is for you to extend your previous code with design patterns, UML, and Java for additional functionality. The design and code you develop here will be used as the starting point for implementing patterns and enhancements in assignment 4 as well. You should use a Java 8 or later environment to develop this code (I recommend Java 17). You may use my example code from assignment 2 for this project, but please clearly indicate in comments if you do.

Overall Assignment – Zoo Simulation Extended

We will continue to simulate the zoo from assignment 2 in this assignment, all functionality previously provided should continue to work. This version of the simulation will introduce several new objects – a new Staff subclass called Vendor, a new building type called a Shop – and some changes in initialization and run behavior.

New animal behaviors: In observing the animals we discovered:

When a Parrot makes a sound, it will say one of five randomly selected phrases.

Felines now have a chance to charge during roaming as Pachyderms do, but only 10% of the time.

Creating Shops and Vendors: During initialization you will want to make five instances of a Shop with the following identities: Gifts, Maps, Drinks, Food, Toys. Each Shop starts with an Inventory of 100 to 200 Items. You should determine the Price of an item for each Shop as an attribute of each Shop (between $1 and $10). Each Shop has a Likelihood of making a sale to zoo visitors (between 10% and 25% as you select). Each Shop initializes to have Cash equal to Current Inventory * (Price / 2). Each shop has a uniquely named instance of a Vendor assigned to the shop.

In a day at the Zoo, all Vendors will perform this new action:

prepareShop: If a Vendor finds that the inventory at their assigned Shop is less than 20 items, they must add 100 items to the inventory. This is paid for by reducing the Shop Cash by (added item count * Price/2). This should be announced as it happens.

The following new action is a Zoo level method. It should occur every day after the zooStatus event:

sellItems: Each day 50 plus a random 0 to 50 additional people will visit the Zoo. (This number should be announced to the console e.g. “Today we have 73 visitors!”) Note that these people do not need to be represented by objects – they do not require individual identity. Each person visiting the zoo will visit each Shop until all Shops are visited or the person exits. (You may determine the order of visiting.) Visitors use the Shop’s Likelihood chance of buying an item from the Shop (reduces Shop Inventory, increases Shop Cash by Price). There is a normal 5% chance after visiting each Shop, that the person will not visit any other Shops – they exit the Shops. The chance of buying an item and of exiting is affected by the Vendor’s SalesBehavior (see below). Note that events from the shops should NOT be announced in console text, the events should be published to the SalesTracker object (see below).

Strategy Pattern: When a Vendor is created, they will have a SalesBehavior set for them (all types should be used at least once). The possible SalesBehavior subclasses are: NoSell, SoftSell, NormalSell, and HardSell.

- A Vendor selling items using NoSell will not impact the Likelihood of a sale or exit.
- A Vendor using SoftSell will increase Likelihood of a sale by 10% with no exit change.
- A Vendor using NormalSell will increase Likelihood of a sale by 20%, with an additional 10% exit chance.
- A Vendor using HardSell will increase Likelihood of a sale by 25%, with an additional 25% exit chance.

Observer Pattern: Create an observer called a SalesTracker. The SalesTracker will subscribe for the following events being published from Shops: A visit, a sale (with price), and an exit. The SalesTracker will use the events to create a summary of the Shop traffic for each Shop for the day. At the zooStatus step, the SalesTracker should use a summary method to display for each Shop: Shop and Vendor name, number of visits, number of sales, total $ sales for the day, number of exits. You may use any observer/subject – pub/sub code approach to implement this behavior.

Singleton Pattern: Create both the SalesTracker and the Hospital as Singleton objects. References to these objects should be taken from a getInstance method from each Class, not via new. Use lazy instantiation for one of the objects, and eager instantiation for the other.

The Zoo simulation will run for 30 days as before. All other Zoo events from Staff will continue to occur and be printed to the console. Display the start and end of each numbered day in the console.

Assignment 2 Part 1 – UML designs for the Zoo Simulation

Using UML tools discussed in class, create the following UML diagrams.

1. A UML Class Diagram – This UML class diagram should include all classes, abstract classes, or interface definitions you feel you should use to design the Zoo simulation. The diagram should be complete, modeling all classes in use in the application. The diagram should clearly indicate any aggregation, association, or inheritance relationships, including any required multiplicity. In class elements of the diagram, identify key attributes and methods, including accessibility. (Usually this does not include constructors.) **Please highlight in the UML diagram where you are implementing Strategy, Singleton, and Observer patterns.**
2. A UML Sequence Diagram – Show the normal initialization and messaging between Shops, SalesTracker, Vendor, and the console in a normal day as a UML sequence diagram. You only need to show normal conditions (no error or alternative paths are needed.)

Please include your name on the submitted materials.

Submission Guidelines – Part 1

I would prefer that the diagrams are made with an electronic drawing tool for clarity. If you must hand draw the diagram (on a whiteboard or on paper) it must be complete and legible. The diagram should be turned in as a PDF. Most tools you use can print their output to a PDF, if necessary, you can break large diagrams into two or more pages. The important part is that what you turn in is readable and remember to put in your name!

Assignment 2 Part 2 – Java Code for the Zoo Simulation

Carefully consider the construction of your Java code based on your OO Design work. Implement the code required in Java to create, initialize, and run the Zoo simulation for 30 days. The code you write for the simulation should identify code related to the three added patterns: **Strategy, Observer, Singleton (including eager and lazy instantiation implementations)**. Clearly comment on at these implementations in your code so that they stand out when code is reviewed.

In addition, all code should be appropriately commented. Any .java files should include a *header block* with the author's name, the class, the program purpose, and the last revision date in the comments. Each *class/interface* and significant *method* in the code should be commented on as well. You do not have to use Javadoc formats for these comments, but you certainly can.

Capture ALL output from the simulation to the console (by cut/paste to a text file or optionally by directly writing to a text file). Also, update the UML class diagram from Part 1 to represent the final outcome of your Java code. Note any changes on the diagram since Part 1 as an annotation on the diagram itself.

If you use code in whole or in part from any external source, say Stack Overflow for example, you must cite your original source and include the URL in your comments. Never submit code copied from another source without providing appropriate citations.

Code in the Java main function should be limited to basic object instantiation and execution. No large blocks of procedural code or logic should be in Java main. All operations and information should be handled between appropriate objects of the solution classes.

Submission Guidelines – Part 2

Submit the following files to CougarVIEW for Part 2:

- any .java files required to run your program
- one text file with captured output from running the program for 30 simulated days
- your updated UML class diagram from Part 1 (as a PDF)

Grading Rubric:

Assignment 2 is worth 80 points total.

Part 1 (UML Diagrams) is 30 points and is due on Friday September 22 at 11:59 PM

- Each diagram is worth 15 points. A thorough design that meets all the requirements for class content and UML annotations will earn full points. Severe misses in UML syntax or incomplete design implementations will be penalized.

Part 2 (Java Code & Results) is worth 50 points and is due on Friday September 29 at 11:59 PM

- (15 points) The code will be reviewed for good OO design – cohesive classes with clear logical connectivity and well named methods. Procedural elements or code duplication will be penalized. Code should be thoroughly commented as requested.
- (10 points) The implementing of the three added patterns: **Strategy, Observer, Singleton (including eager and lazy instantiation implementations)** should be clearly identified in comments.

- (15 points) The output from code execution should show all expected object actions and flow including new behaviors and objects. Missing elements will be penalized.
- (10 points) The updated UML Class Diagram should clearly show any changes since part 1, and an annotation on the drawing should describe the major changes. If no changes were made, this should be clearly annotated on the drawing as well.

Overall Project Guidelines

Assignments will be accepted late per the published policy from the syllabus:

- Up to two days late from posted submission date/time: -5% grade penalty
- Up to an additional three days late from posted submission date/time: -15% grade penalty
- Assignments will not be accepted after 5 days from posted submission date/time

Use office/student hours and e-mail to reach the instructor regarding homework/project questions, or if you have issues in completing the assignment for any reason.